# MULTIPLATFORM JAVA GATEWAY

Maros Balint

Bachelor's Thesis

May 2011

Degree Program in Information Technology

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

Author(s)

BALINT Maros

Type of publication
Bachelor´s Thesis

Date
16.05.2011

Pages
51

Language
English

Confidential

( ) Until

Permission for web publication
( X )

Title

MULTIPLATFORM JAVA GATEWAY

Degree Programme

Information Technology

Tutor(s)

MIESKOLAINEN Matti

Assigned by

Ixonos Finland Ltd.

Abstract

The Java Gateway was a part of the internal project for the Ixonos Company. The purpose of the project was to show demonstrative programs to representatives of the company. The Java Gateway was a centre part of the system connecting clients with servers running on the different platforms. Its purpose was to accept multiple connections and linked proper pairs of the client and the server. Various types of messages are sent trough the gateway, for example numbers, strings and files.

This thesis details the development process of the Java Gateway, creating the communication protocol and explaining messaging in various different platforms. The testing was an important part of the work.

In the first part are explained reasons and motivations to start a new project. Tasks are discussed. In the following chapter is development environment presented. Furthermore is explained the important theory. Afterwards is illustrated a development process of the Java Gateway and its separate modules. A designing of the communication protocol is described. Later on is shown handling of the various types of messages in other platforms like in Android, Qt or Python. Finally, the Gateway is tested and the results and conclusion are presented.

The result of the work is the functional software able to achieve all required tasks. Demonstrative programs were shown with the success to representatives of the company. Whole internal project can be transferred to the real customer project in the future. In this case a further developing would be required.

Keywords

Java, gateway, Android, Qt, Python, message, interface, sending, receiving

Miscellaneous

# CONTENTS

# FIGURES

# 1 INTRODUCTION

The internship in the Ixonos company started with making a small demo that consisted of the Java server that was able to accept just one client and its purpose was just to process events. A concept like this is quite limited and a more complex solution was soon searched.

The whole project was divided into more parts consisting of different types of clients running on different platforms. The first group of clients sends messages that should be handled by the server clients. (From the author's point of view, everyone is a client). An opposite direction of sending messages is also possible. Java gateway (further referred to as GW) is logically situated between the first type of clients (further referred to as MT) and the server clients (further referred to as SC). Thus mobile clients and SC are not directly connected. Instead they connect through GW. The following paragraph discusses the need for the extra part and the possibility of connecting them directly.

There is an opportunity to connect multiple mobile clients to the one server client. Clients can run on different platforms, for example: Android, Symbian, Maemo, MeeGo, Windows mobile OS, even notebooks or any other mobile devices. SC can also run on different platforms. To make these two parts universal and independent would be very difficult without any mediator. If using one SC should be changed by another, there is need to disconnect all MT from SC and put another IP and connect again. Another disadvantage for SC would be the handling of multiple connections from MT. Thanks to the GW server, a client needs to handle just one connection with the gateway. If SC died, all mobile clients would lose a connection. GW can provide handling of this situation. It can redirect MT to backup SC or just tell them that the SC is not responding.

As can be seen now, GW server is a very important part in our project. It connects all different types of mobile clients with SC. It routes events between correct pairs and handles all situations that can happen. GW must be stable, reliable, fast and able to process different types of information. The security level of the whole application also increases. SC is connected to the gateway and it can receive only messages that were filtered by GW, no unknown or dangerous messages from pirate's applications.

The present thesis discusses a Scrum agile software development, the phases of developing the Java gateway server by showing examples of receiving and sending messages on different platforms. Future development of this project and at the end conclusion and results are discussed.

# 2 OBJECTIVES OF THE THESIS

## 2.1 Reasons and motivations for a new project

The objective of the thesis was to make separate modules that can be easily replaced without need to changes other modules: the program was separated into more logical packages. Inside every package there are classes which cooperate together. Inheritance and its advantages as well as threads are necessary.

All mentioned techniques have to be combined in one functional piece of software. To achieve this, Scrums and planning is needed. This project requires also regular every day communication with colleagues to design a common communication protocol and common rules. It also requires complete exception handling and all possible failures have to be counted with. GW cannot crash at any circumstances because it would mean disconnection of all clients and the whole system would be out of service. Other platforms are discussed and reading and sending data in the concrete platform is implemented. For example, asynchronous reading of data in an Android device requires a thread that reads the incoming messages all the time and puts them to the message handling management.

## 2.2 Objectives of the project

The main goal is to make functional application in reasonable time, which does not require the highest quality of the software code. All security aspects are left for the future development. The whole design and implementation of the GW was left for the author of this thesis. At the end of each sprint a demonstration was shown which revealed the progress of the application. This concept of work was very effective and the first results came early. Therefore, the main objective was to be able to present a functional demonstration with the required abilities. GW part of the application is the core issue. When GW works properly finishing other parts can be focused on.

## 2.3 Project tasks

The Scrum meetings all tasks needed to be accomplished were discussed and they were ordered by the importance. This approach secured that the most important tasks were accomplished first.

The first task was to set up a testing environment and Git repositories for back up. This is discussed in the next chapter in more detail.

The second and the hardest task was to design a skeleton of the GW. In these days it is very rare to start an implementation of the project from scratch. This is usually the most important part of the whole planning because if started in a wrong way it can be extremely difficult to rebuild the application in the future.

The next task was to establish a connection with a client and send an integer message to GW and back. In this phase the hierarchy of the messages had to be resolved. Afterwards the input / output module should be developed. It means creating a separate independent module the purpose of which is to accept more connections and secure constant reading of incoming messages. This part also needed to be tested. Another task included connecting and disconnecting clients, considering what kind of message should be send and what should happen on GW.

Later in the process the question of session handling and routing messages between clients appeared, which means creating a next separate module the purpose of which is to correctly route messages between multiple clients and to provide queues for incoming messages.

Following types of messages could be added after finishing the previous tasks. The first task was to find out how to send strings between clients. This task was more difficult especially on the Qt part. The last and the most difficult quest was to make the sending files. Then it could be found out how to make constant reading of messages in Android possible without freezing an application.

# 3 DEVELOPMENT ENVIRONMENT

## 3.1 Java gateway

At first JDK (Java Developing Kit) needed to be installed. JDK includes also JRE (Java Runtime Environment) needed for running a Java application. The latest version of JDK is downloadable on the following webpage: http://www.oracle.com/technetwork/java/javase/downloads/index.html

Eclipse Helios was used as development environment for GW. Based on experience this environment can be suggested for similar tasks. Eclipse can be downloaded on page: http://www.eclipse.org/downloads/ . Another option can be NetBeans or some other environment and this is basically all that needed to be done for GW.

For developing an Android application also JDK was needed. The second important issue was to download the latest Android SDK on the following web page: http://developer.android.com/sdk/index.html.

From the package it was simply unzipped and installed. Based on previous experience it was known that it is better to download an older version of Eclipse called Galileo, which can be found at the following site: http://www.eclipse.org/downloads/packages/release/galileo/sr2.

Then Android plugin needed to be installed into Eclipse. The instructions to do this are as follows: Select Help > Install New Software…In the Available Software dialog, click Add…In the Add Site dialog that appears, enter a name for the remote site – for example: “Android Plugin” Add the following URL to Location field: https://dl-ssl.google.com/android/eclipse/ .

The final step was to get ADT functional inside Eclipse. This could be done as follows: Select Window menu and choose Preferences. Select Android node and set the SDK location field to path of the Android SDK. After this procedure our Android **developing** environment was ready to use.

## 3.2 Qt framework

In this project Qt applications were developed in the Linux Ubuntu operating system. Gcc compiler needed to be installed as well as l Qt SDK including Qt Creator environment. It could be found at the following site (32b version): http://qt.nokia.com/downloads/sdk-linux-x11-32bit-cpp . After installing Qt applications could be developed. Qt libraries and Qt Creator are possible to install also in Windows, but based on the author's experience it works faster and smoother on Linux.

## 3.3 Python

Python application was also developed in Linux Ubuntu OS. The base version of the Python compiler should be by default installed. Then a basic application could be made in "gedit" and run through the terminal, for example: "python main.py". The users need to be in the folder where the application is. Pydev plugin in Eclipse can also be used here.

# 4 AGILE SOFTWARE DEVELOPMENT – SCRUM

In this chapter basic theory about Scrum is shown. Scrum methodology is used in Finnish IT companies and it is a relatively new and modern way of software development. Advantages of this methodology are described in the next chapter.

## 4.1 Why to choose agile methodology

In the traditional approach to software development, a product specification was first made. This phase is called a requirements specification. Then whole system was designed, class by class, method by method. After that the program was implemented, tested and installed for the customer. Last and the longest part was the maintenance of the software. This methodology is called a waterfall -model. This approach is still used in many companies. The advantage of this approach is detailed planning, continuing step by step, however, the major disadvantage is its weak flexibility. If a customer decides to make major changes or design faults are found in the late phases, all steps might need to be run again. This can cause delay and is likely to be expensive. In these days customer aspirations change very quickly and the whole development is still in progress.

Another advantage of this approach is that functional software is available during the whole process. A customer can see the progress and decide better what he/she actually wants. He/She can make changes in the time when the actual part of software is being developed and the team can easier implement them.

## 4.2 Basic terminology of the Agile Scrum development

Although the Scrum approach was originally suggested for managing product development projects, its use has focused on the management of software development projects, and it can be used to run software maintenance teams or as a general project/program management approach. It was designed to increase speed and flexibility of the production. The most important roles in the Scrum can be defined as follows.

**Product Owner:** represents the customer side of the project. He/She writes user stories and makes a product back log. He/She can prioritize all tasks and he/she keeps watching if the team meets all requirements.

**Scrum Master:** his/her role is to watch about rules. He/She acts like a mediator between the product owner and the team. He protects the team from outside influence and takes care of all necessary equipment. Scrum master also leads Scrum meetings.

**Team**: The Team is responsible for making the project. The team usually consists of three to ten people. The Team is usually self-organized and members of the team choose the tasks to make. They design, implement and usually also test the application.

Some terminology used in Scrum is explained as follows. The production of the whole application is divided into **Sprint**s. A sprint can last seven to 30 days. The product owner makes the **Product Backlog** which is actually the list of all wishes to be made. It is a specification of the program. A very important part of the Scrum is the **Sprint Planning Meeting.** It usually lasts four to eight hours and it is at the beginning of the each sprint. First hours the product owner prioritizes with the team product backlog. Then the team selects what work is to be done. They prepare a new **Sprint Backlog.** It is list of tasks that should be resolved in the actual sprint. Then the team manages all implementation and the product owner cannot change the sprint backlog or influence the team. Every day there is a short **daily Scrum** session, which lasts max. 15 minutes. The team and the Scrum master discuss what was done previous day, what problems appeared and what is going to be made. If some problems need more time to resolve, they are discussed after the daily Scrum. It can happen that the team finds an unresolved problem or barrier that cannot be resolved. In this case the sprint can be interrupted and a new sprint planning meeting takes place. The sprint can be interrupted also if some new important changes from the product owner occur. At the end of each sprint there is the **Sprint review meeting**. All progress the team made is discussed and a demonstration program is shown. This is strong motivation for the team to do a good job.

As can be seen, this is a very flexible and dynamic methodology that allows changes during the developing software. Team members are more independent and skillful in

designing and testing. This methodology is motivating to programmers and they work faster.

# 5 THEORETICAL BACKGROUND

## 5.1 Data structures

“In computer science, a data structure is a particular way of storing and organizing data in a computer so that it can be used efficiently. Different kinds of data structures are suited to different kinds of applications, and some are highly specialized to specific tasks. For example if some item should be found quickly, a hash table can be used. Or, if an item with the highest priority is often needed, a heap is used that is a particularly ordered structure where an item with the highest priority is at the beginning”.

As can be seen there are many types of data structures that can be used in specific situations. The data structures used in this thesis project are described as follows. (http://en.wikipedia.org/wiki/Data_structure)

### 5.1.1 Queue

A data structure queue is taken from real life. The order of “entities” in the computer terminology is kept. For this data structure an acronym FIFO (first-in-first-out) is often used. This is equivalent to the requirement that once an element is added, all elements that were added before have to be removed before the new element can be invoked. “A queue is an example of a linear data structure. Queues provide services in computer science, transport, and operations research where various entities such as data, objects, persons, or events are stored and held to be processed later. In these contexts, the queue performs the function of a buffer. Queues are common in computer programs, where they are implemented as data structures coupled with access routines, as an abstract data structure or in object-oriented languages as classes”. Common implementations are circular buffers and linked lists. (http://en.wikipedia.org/wiki/Queue_%28data_structure%29)

Basic operations with queue are listed as follows:

- **is empty**: returns true if the queue is empty or false if is not
- **front**: return a reference to the first element without removing it
- **pop**: get first element and delete it from the queue
- **push**: add a new element to the back of the front

- **size**: returns number of elements inside queue

In Java ready collections to perform these operations can be used. A queue is a linked list.

### 5.1.2 Map

A map is a very useful data structure which is assigned some type of data with a specific and unique key. In this thesis it is used to keep clients and their threads together and it allows a quick access to the concrete client.

"In computer science, a hash table or **hash map** is a data structure that uses a hash function to map identifying values, known as keys (e.g. a person's name), to their associated values (e.g. their telephone number). Thus, a hash table implements an associative array. The hash function is used to transform the key into the index (the hash) of an array element (the slot or bucket) where the corresponding value is to be sought". (http://en.wikipedia.org/wiki/Hash_table)

A hash map is very often used because of its extremely fast access to items. This time depends on how well a hash function is implemented for a concrete situation; however, the access time is close to O(1). It means that with one calculation it is possible to access to the concrete item.

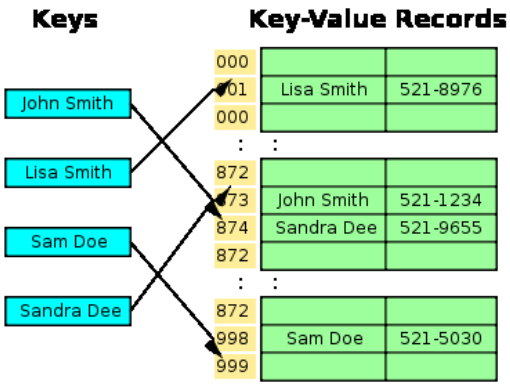


FIGURE 1. An example of hash table

Typical operations can be:

- **contains key**: returns true if a map contains inserted key or false if not
- **put**: add a new key and value into the map
- **remove**: delete an item with the given key

### 5.1.3 Set

"In computer science, a set is an abstract data structure that can store certain values, without any particular order, and no repeated values. It is a computer implementation of the mathematical concept of a finite set. A set can be implemented in many ways. For example, one can use a list, ignoring the order of the elements and taking care to avoid repeated values. Sets are often implemented using various flavors of trees, tries, hash tables, and more". In this project case a hash set in Java was used to avoid duplicity. (http://en.wikipedia.org/wiki/Set_%28computer_science%29)

The operations are listed below:

- **add**: insert a new value into the set if there is not same value already
- **remove**: delete the element from the set if it is there
- **size**: returns number of elements

## 5.2 Threads

"A **process** is an instance of a computer program that is being executed. It contains the program code and its current activity. A thread of execution is the smallest unit of processing that can be scheduled by an operating system. The implementation of threads and processes differs from one operating system to another, but in most cases, a thread is contained inside a process. Multiple threads can exist within the same process and share resources such as memory, while different processes do not share these resources. In particular, the thread of a process share the latter's instructions (its code) and its context (the values that its variables reference at any given moment).

On a single processor, multithreading generally occurs by time-division multiplexing. The processor switches between different threads. This context switching generally happens frequently enough that the user perceives the threads or tasks as running at the same time. On a multiprocessor or multi-core system, the threads or tasks will actually run at the same time, with each processor or core running a particular thread or task". (http://en.wikipedia.org/wiki/Thread_%28computer_science%29) In the next figure is shown the difference between execution on a single-core and dual-core machine. Correct using of threads can significantly accelerate the running of the program. Threads are needed also in this project application. E.g. a thread is required for continuously waiting for accepting new connections.

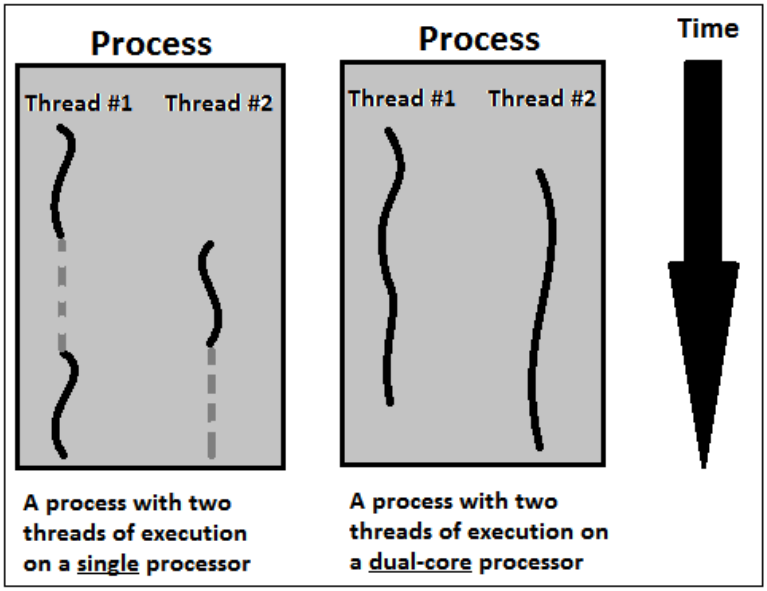


FIGURE 2. Threading on single and dual core processors

## 5.3 Socket

"An Internet socket or network socket is an endpoint of a bidirectional inter-process communication flow across an Internet Protocol-based computer network, such as the Internet. The term Internet sockets is also used as a name for an application programming interface (API) for the TCP/IP protocol stack, usually provided by the operating system. Internet sockets constitute a mechanism for delivering incoming data packets to the appropriate application process or thread, based on a combination of local and remote IP addresses and port numbers. Each socket is mapped by the operating system to a communicating application process or thread". (http://en.wikipedia.org/wiki/Internet_socket)

"A socket address is the combination of an IP address and a port. Computer processes that provide application services are called servers, and create sockets on startup that are in listening state. These sockets are waiting for initiatives from client programs. Communicating local and remote sockets are called socket pairs. Each socket pair is described by a unique 4-tuple consisting of source and destination IP addresses and port numbers, i.e. of local and remote socket addresses. Thanks to that an OS can distinguish multiple clients' sockets connected to the same server's IP and port". (http://en.wikipedia.org/wiki/Internet_socket)

# 6 DESIGN

## 6.1 Explaining the concept

The role of the GW is to accept connections from all kind of clients that send proper open session messages. GW does not bother what type the client is or on which platform it runs. GW provides routing of messages between clients who want to communicate together.

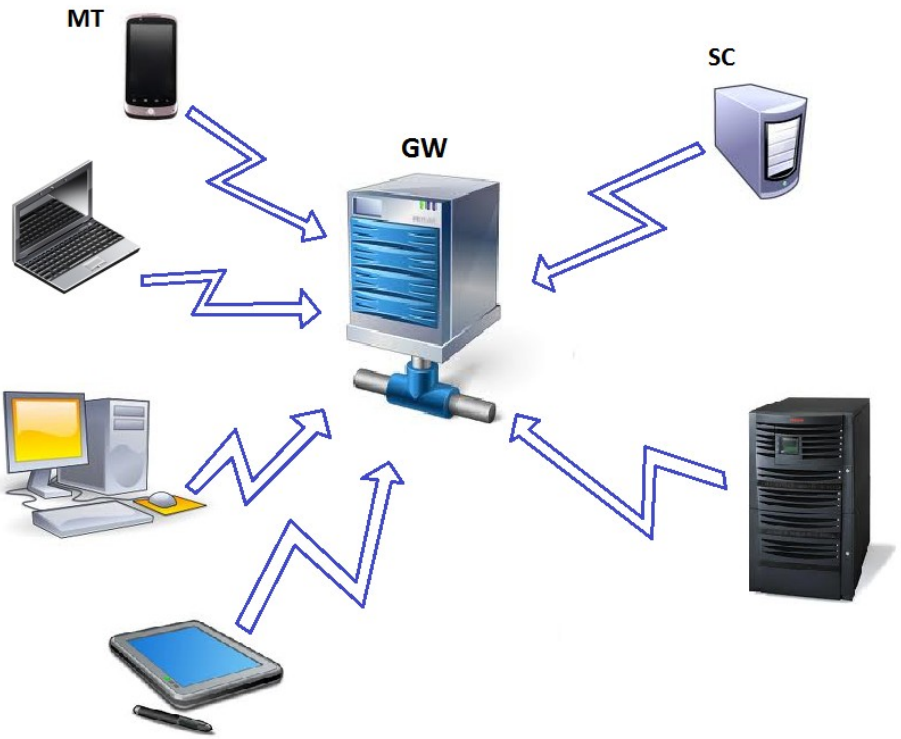


FIGURE 3. Basic concept of GW

The basic idea of GW is illustrated in figure 3 and how it is used to connect clients on the left side with servers on the right side. It can be possible; however, if needed also to connect together for example two mobile clients. GW must accept multiple connections, therefore the first step should be to establish connection with some client.

## 6.2 Phase one, (trying to) establishing connection

Sockets were used for transferring data in the network. Data is transferred via TCP packets. TCP protocol ensures the reliable delivery of the data. The first program was really simple. Its only purpose is to accept connection, receive some data and send them back.

An example of the server side code can be seen below:

```
1  ServerSocket ssocket;
2  ssocket = new ServerSocket(this.port); //port is int value, f.g. 50000
3  Socket s = ssocket.accept(); //waiting until client requires connection
4  sock_in = s.getInputStream(); //get input stream
5  sock_out = s.getOutputStream(); //get output stream
6  int i = sock_in.read(); //read 1 byte
7  sock_out.write(i); //send it back
```

It is basically very simple code. On the second line a new server socket is created with the specified port. Then the program waits until some connection from the client is established. Afterwards the streams are initialized and it is possible to receive and send some bytes. Method “flush” is needed. The client side code example is illustrated as follows:

```
1  Socket s;
2  InetAddress ia; //represents an Internet Protocol (IP) address
3  ia = InetAddress.getByName(null); //local host, it can be ip also inserted
4  s = new Socket(ia,port); //connect to the server
5  OutputStream out = s.getOutputStream(); //get streams
6  InputStream in = s.getInputStream();
7  out.write(1); //write byte to the socket with value 1
8  out.flush(); //flush content
9  int response = in.read(); //read response form the server
```

The testing started from the very beginning. In this simple example a server part of the code and also a client part are used. This can be another class with “main” method in the same project. This will be very helpful in the next phases of the developing.

## 6.3 Sending and receiving 32b integers

Just one byte was sent in the previous example. It refers to a number with maximum value 255 if it is unsigned integer. Of course this is not enough in the most cases. The next question was how to send a bigger value and ensure that this value will be the same in all platforms. One can assume that a number is just a number and it is the same anywhere but it is not. There was a situation that in Qt C++ the value was totally different. It was decided that 32b integer should be enough in most cases. It represents a max value 4 294 967 295. The basic idea is to divide the whole number into four bytes and send them one by one to the socket. The same idea goes for the receiving: receive four bytes one by one and put them together. The next side has to know how to rebuild or fragment that number.

An example of writing 32b integer into socket is illustrated below:

```
1   public static void writeInt32b(OutputStream out, int value) throws IOException
2   {
3           out.write((value >>> 24) & 0xFF);
4           out.write((value >>> 16) & 0xFF);
5           out.write((value >>> 8) & 0xFF);
6           out.write((value) & 0xFF);
7   }
```

This method has two incoming parameters, the value and the output stream are to be sent. It is a static method which means that it can be used anywhere in the program without making an instance of the class where this method is situated. To achieve fragmentation into four bytes bitwise and shift operators were needed. Bitwise operator n >>> p means as follows: shifts the bits of n right p positions. Zeros are shifted into the high-order positions. For example, 256 >>> 8 = 1. In the binary it looks like: $111111111_2$ >>> cut 8 bites = $1_2$. A symbol '&' is binary AND. It is necessary for example in this case: the number is one million. In the line 5 a right shift is made. $1000000_{10}$ >>> $8_{10} = 3906_{10}$. The result in binary is $111101000010_2$ which is more than 1 byte which should be sent. Thus, binary AND is used to cut the upper bits. $111101000010_2$ & $11111111_2 = 01000010_2 = 66_{10}$.

The receiving method is presented as follows:

```
1   public static int readInt32b(InputStream in) throws IOException
2   {
3           int a = in.read(); //read step by step 4 bytes
4           int b = in.read();
5           int c = in.read();
6           int d = in.read();
7
8           if ((a < 0) || (b < 0)|| (c < 0)|| (d < 0))
9                   return -1; //in case that something fails
10
11          return (a << 24)|(b << 16)|(c << 8)|(d);
12  }
```

Like the previous method also this one is static. Four bytes are read into variables and a 32b integer is returned back. In this case left shift operator and binary operator OR are used. For example, if number $1000_{10}$ should be received, upper two bytes contain zeros and lower contains $3_{10}$ and $232_{10}$. In binary:

```
1   00000000 00000000 00000000 00000000 OR
2   00000000 00000000 00000000 00000000 OR
3   00000000 00000000 00000011 00000000 OR
4   00000000 00000000 00000000 11101000 =
5   00000000 00000000 00000011 11101000
```

## 6.4 Creating messages hierarchy

If just a number is sent every time without any meaning, all communication is useless. Combination of numbers is used to tell more to the other side. Therefore, the **communication protocol** should be defined It means to define which number has what meaning and how many numbers or other types the concrete message includes. It can be defined that the interface contains constants. Every constant represents a new type of message. For example, GW receives number 1 and looks for the constants and sees that this is an open session message and it can expect next two values that represent some needed attributes. If GW receives a value that is not in the list of the constants, it can disconnect a client because the communication protocol was broken.

As can be seen, every message can have more attributes while the number of parameters is also variable. To create new class for every different message looks like a good solution. Various numbers of attributes can be defined with a different meaning.

It is also clear that every message has to have some basic functionality like write a number or read an integer from the socket. Everyone has also one attribute named type of the message. Thus the inheritance looks like a good solution to save some lines of code and we can also use some abstract methods. It means that child class has to re-implement abstract methods from the parent and it is ensured that the expected functionality must be implemented.

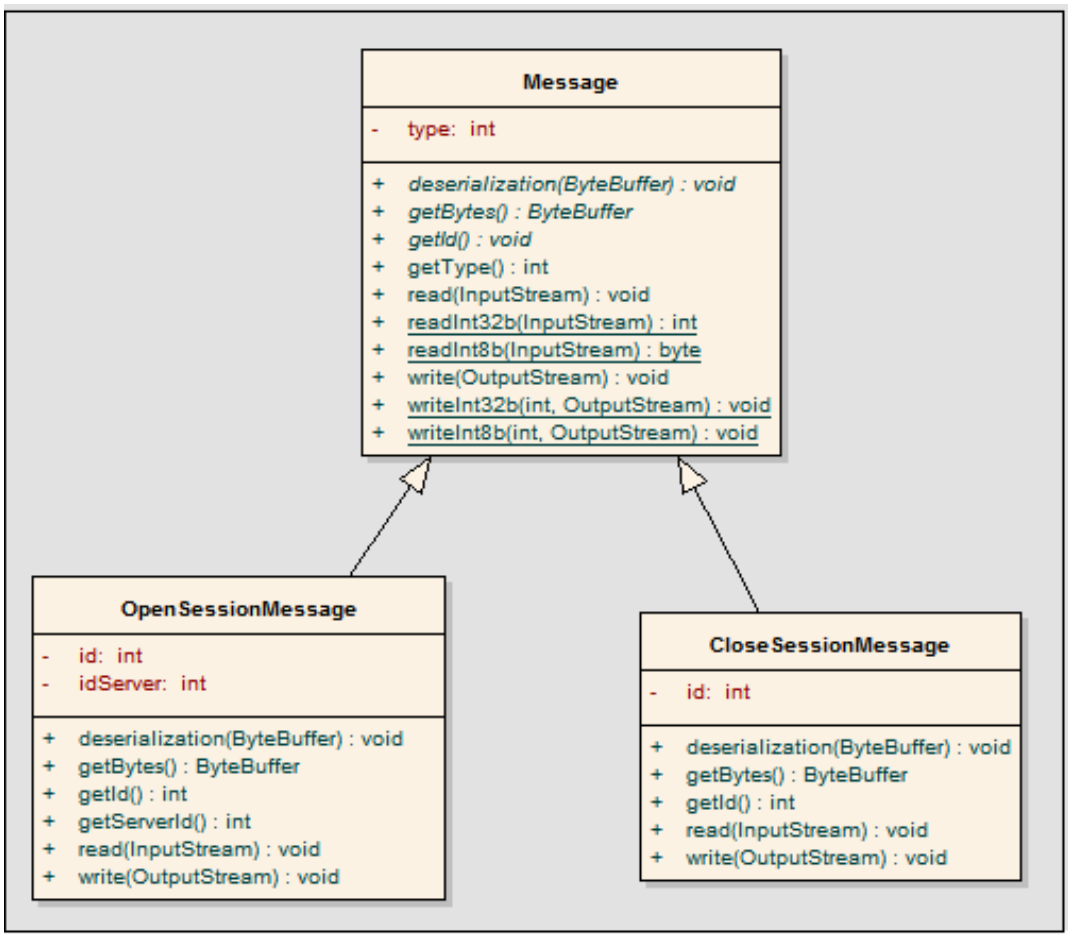


FIGURE 4. UML diagram of messages

The figure above illustrates that every new “MessageType” class has to inherit from the “Message” class. It gives them direct access to static methods like “writeInt32b” or “readInt32b”. They can override write and read method and they have to re-implement methods “getBytes, deserialization and getId”. Their purpose is explained later. When creating an instance from inherited classes super constructor of the “Message” class has to be used that defines the type of the message, which ensures that every message has to have a defined type. The “Message” class is abstract and therefore no instance can be made from it. Another advantage of the inheritance can be used and it is that child can be assigned to the parent, e.g. “Message = OpenSessionMessage”. It is useful when a method with parameter of some “Message” class is needed. There is no need to create a method for every one new “MessageType” class but one method with parameter “Message” can be used, e.g. “onMessageCome(Message m)” and in place of ‘m’ can be any child class of the parent “Message”. It has also “getType” method that returns the correct type of the actual message.

## 6.5 Developing input / output module

It is possible now to receive and send various messages but only with one client. A module is need that is responsible for socket handling with many clients. This module should be easily replaced or changed without any code modifications in other modules. The best way to do this is to create interfaces. IO module has to implement that interface and outside modules know just about the interface between them and they do not have to bother if the device IO read from socket or from file or similar.

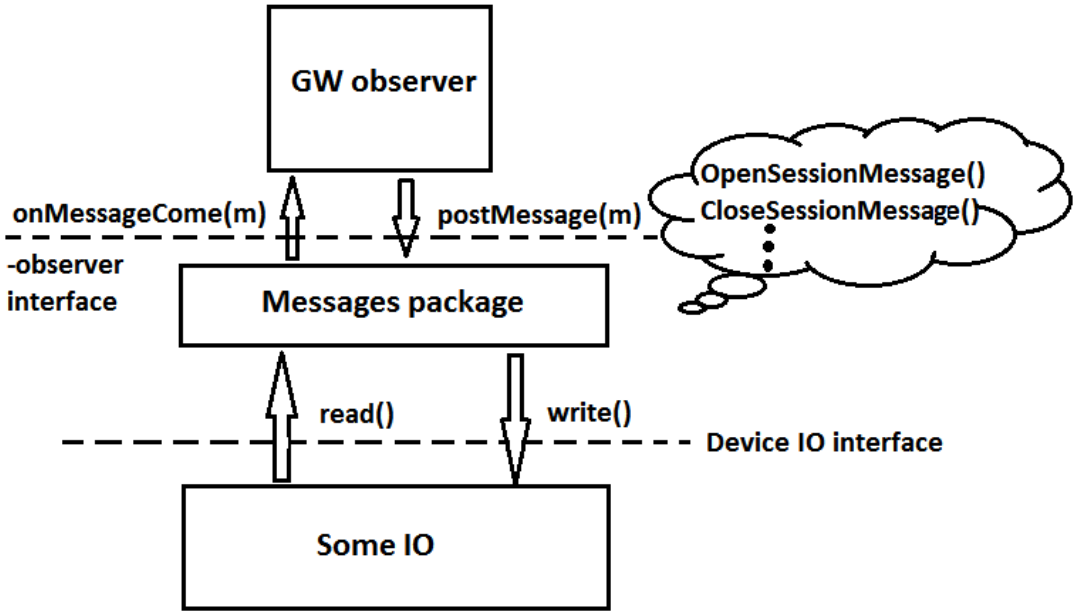


FIGURE 5. Input / output device interfaces

In figure a basic concept of how it should look can be seen. Between Some IO and messages there is an interface containing methods read and write bytes from the stream. Actually it should not be important what kind of data was received. Between GW observer and Device IO there is the interface that contains the method "onMessageCome". Basically this interface and the method "postMessage" are only one facility able to communicate with the outside world. Thanks to that, it is possible to develop one program with more developers. They just need to agree about the interfaces. An example of the interface can be seen below.

```
1  public interface DeviceIOObserver {
2      public void onMessageCome(Message m);
3  }
```

### 6.5.1 Implementing GW observer

Simple “MessagesGW” class can be started with. It implements the previous interface. It means that this class has to implement “onMessageCome”. At first it just sends the incoming messages back.

```
1   @Override
2   public void onMessageCome(Message m) {
3       deviceIO.postMessage(m);
4   }
```

On line 3 can be seen “deviceIO” variable. It represents “ADeviceIOImpl” class. This class is a very important class and it is explained in the next chapter.

### 6.5.2 Implementing “ADeviceIOImpl” class

The purpose of this class is to create a server socket and listen for new connections. After accepting a new connection a new thread is created which handles the socket communication with the concrete client. It is very important that the handling of the server socket is taken care of by only one class and one instance of it. Because of that a **singleton design pattern** can be used. The singleton allows creating just one instance of the class and this instance is available from whole program.

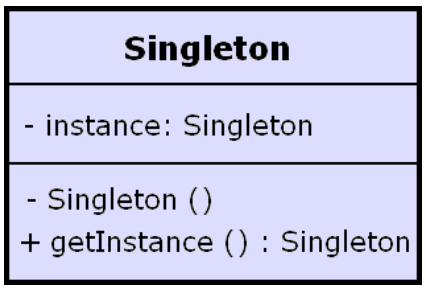


FIGURE 6. A singleton example

A constructor of this class is private, so it is impossible to create an instance from outside. A method “**getInstance**” is public static, so any class can get instance of this class. If the instance is NULL a private constructor is called. So GW observer gets an instance of this class and calls method “**init**” with the parameter GW observer. In the method “init” a new server socket is initialized.

It was already mentioned previously that there is need to wait at this line of code: *Socket s = ssocket.accept();* The application is frozen until a new connection is

accepted. Because of that a thread is needed that can wait for a new connection continuously and theoretically infinity time. In the next public method “start” a new thread is created. When a new connection is accepted a new thread is created that is basically a new class “ClientThread”. An observer parameter is passed to it and in the constructor of this class method “start()” is called that basically calls “run()” method. This class is discussed later in more detail. This attitude to make a new thread for every new client is not the best. It is useless for few hundred or thousands clients, however, this program was made for demonstrative purposes, therefore usually for few clients.

A method “**postMessage(m)**” at this time sends the incoming message to the socket of the client. How to handle more clients is discussed later. A routing of messages will be needed. Thanks to another advantage of the objective programming named polymorphism it is possible to do this:

```
1  public void postMessage(Message m){
2      m.write(sock_out);
3      sock_out.flush();
4  }
```

It is a “write” method called of the Message. Actually it is called the proper write method overridden by the child.

### 6.5.3 Implementing “ClientThread” class

It inherits from the Thread class. Calling method “start” a method “run” is called. It is passed by the constructor parameters like socket, observer and client’s ID. Then the output and input streams are initialized. Method “start” is called and it is sent to the client and his/her ID is generated in GW. Its purpose is needed in routing messages.

Inside “run” method is an infinite while a loop is made. It is all the time reading incoming messages. At first the type is read. Then the switch and case block comparing if the type is equal to some known type from the “MessagesIDConstantsInterface”. If not, the loop is interrupted and a client is disconnected because he/she broke the communication protocol. He/She used an unknown type. If the type is correct a new message class is created and its “read” method is used. An example of this is shown below.

```
switch (type) {
  case MessagesIDConstantsInterface.CLOSE_SESSION:
   CloseSessionMessage cs = new CloseSessionMessage();
   cs.read(sock_in);
   observer.onMessageCome(cs);
   break;
}
```

As can be seen method from “MessagesGW” can be called directly. The method “onMessageCome” is part of the interface. The whole pointer of the “MessagesGW” class was not passed, only the “DeviceIOObserver”. This means that only a method from that interface can be used and no other public method from the “MessagesGW” class. Later on it needs more logic handle situations like disconnecting or lost connection. The figure below presents the whole GW functionality programmed until now.

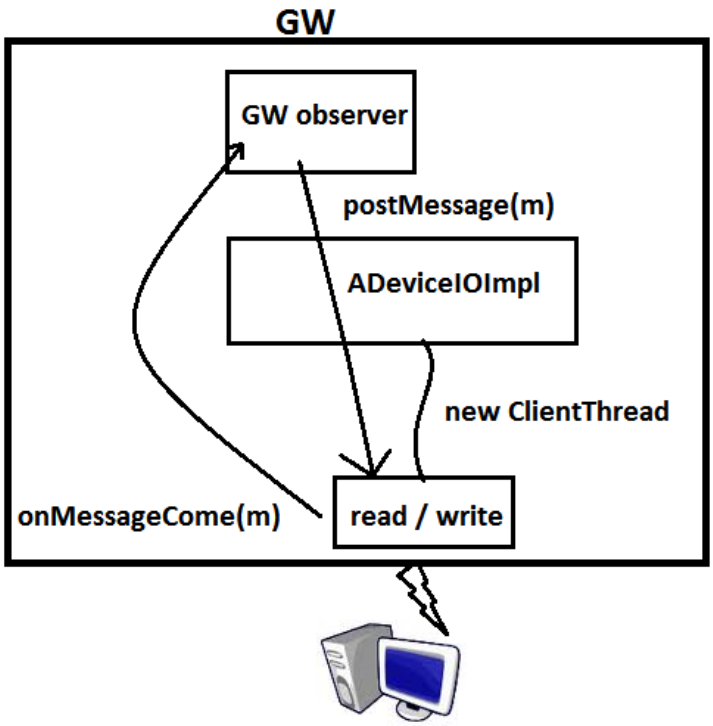


FIGURE 7. GW with device IO, first version

## 6.6 Developing of the session management

One connection receiving and sending some numeric messages can now be accepted. How about accepting more connections and requiring routing between specified clients? A new module has to be built that is responsible for routing messages and handling specific situations like opening sessions and closing them. It should also be

independent and changes on it should not require code changes outside the module. Modifications on IO device are also necessary. Figure 8 illustrates an interface between the observer and the session manager. There is illustrated an event handling and methods used to put and poll messages to proper queues.

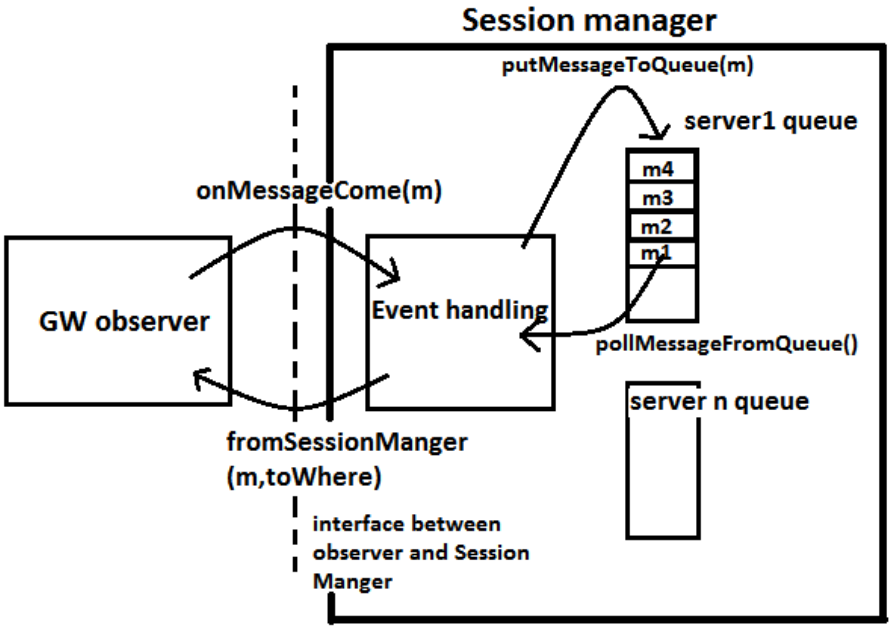


FIGURE 8. Concept of the session management module

The "DeviceIOObserver" interface can be added a new method "fromSessionManager" this is the only output method from the whole module. A new class needs to be created that is responsible for messages handling and routing. One of the tasks was to create an incoming queue for the messages. Later it should be possible that servers will send the request to fetch all messages in the queue with single function call.

### 6.6.1 Changes on "ADeviceIOImpl" class

It is possible to accept just one connection now. The pointer to the new thread is stored in the class and an access to it is direct. Now there is need to accept more connections and it is impossible to have a direct access to all new threads created. Therefore a new map of clients is created where new classes are stored. As a key client ID is used which is unique for every new connection. A method "postMessage" has one more parameter called "toWhere". It is used to find in the map a correct class to handle the socket connection with the concrete client. A new method

"sendToSocket" was made to send the message to the right socket. Below is an example of accepting a connection and putting a new client thread class into the map.

```
1  private Map<Integer, ClientThread> mapOfClients; //key: clientID
2  ...
3  mapOfClients = new HashMap<Integer, ClientThread>();
4  while(true) {
5          ClientThread client;
6          Socket s = ssocket.accept();
7          try {
8              client = new ClientThread(s, observer, clientIDs);
9              //put to the map client and his id
10             mapOfClients.put(client.getClientID(), client);
11             clientIDs++; //ID automatically incremented
12         } catch(IOException e) {
13           // If it fails, close the socket,
14           s.close();
15         }
16 }
```

### 6.6.2 A "SessionManagement" class

There are many possibilities to link correct pairs and send a message to the correct client or server. One option can be to send in every message two attributes, from whom the message coming and to where it is to be sent. That would be very easy to handle but an overflow socket communication and sending unnecessary bytes should be avoided.

The second option can be to send only the client id with every message. A server id has to be sent only in open session messages. In this class information can be stored which client is communicating with which server. When a message comes, client id can be checked, a look to the map is taken. A key is the client's id and the value is the id of the server. Then a method "fromSessionManager" is called that sends a message to the observer and from observer to the device IO.

To achieve this two maps were created:

```
1  //map of sessions, key: ClientID, value: server ID
2  private Map<Integer, Integer> sessions;
3  //map of Servers, key: server ID
4  private Map<Integer, Servers> servers;
```

When a server sends an **open session message** a new instance of the "Servers" class is created. It contains some useful information like a queue for the incoming messages. An open session message from the client contains its id and the id of the server it

would like to communicate. It is put to the sessions map. This step presents actually the whole linking of the communication between the client (MT) and server (SC). When some other message comes from the client, just a look at the sessions map is taken and a message is sent to the correct server. In device IO it is a map of threads handling sockets and thanks to the parameter “toWhere” a message is sent to the correct client or server.

Servers have only their own messages for opening and closing a session with GW. In this case inside message is their own id sent. It was decided that servers know their IDs. Thus after accepting a connection the server has to send an open session message containing its id. Then it is put to correct maps. In any other case it sends the client’s id inside a message. Because of that a new parameter was added to the method “onMessageCome” named “fromWho”. It is easy to find out from who a message is coming because  it is know from which thread and socket it is coming.

### 6.6.3 A “Servers” class

This class stored important information about each server like its ID, clients who communicate with it and a queue for incoming messages. Here data structures queue and set were used. In the code example below is shown declaration and initialization of these structures in Java language.

```
1   private int id; // id of server
2   private Queue<Message> incomingQueue; //queue of messages
3   private Set<Integer> clients; //set of clients communicating with this server
4   //constructor
5   public Servers(ServerOpenSessionMessage os){ //open session message like a parameter
6       id = os.getId(); //get ID of server from the open session message
7       incomingQueue = new LinkedList<Message>();
8       clients = new HashSet<Integer>(); //no duplicates allowed
9   }
```

This class also contains methods for putting and fetching from the queue and for inserting new clients into the set. There is also a method for getting a whole set of clients because this set is used when the server sends close session message.

### 6.6.4 Handling a close session message

Another very important message is a close session message. When it comes from the client it is an easy job to do. The client is removed from the map and its close session message is also sent to the server. It is also removed from the set inside “Servers” class. The same goes for Device IO, it just removes a client from the map and closes

the socket communication. If a server sends some message while a client is disconnecting, this message is just discarded. What happens if the server (SC) sends it and there are still some clients connected to him (through the gateway of course)?

It is important to let clients know that a server is shut down or it crashed. Nobody likes an application that crashes without any explanation. If a server somehow fails, a close session message is generated and sent to the session manager. Thus a session manager actually does not have to know what exactly happened because it receives a close session message anyway. Now it is time to use a set inside the "Servers" class.

```
1  case ServerConstantsInteface.CLOSE_SESSION:
2      ServerCloseSessionMessage cs = new ServerCloseSessionMessage();
3      //create back a close session message from message
4      cs.deserialization(m.getBytes());
5      //get set of clients
6      Set<Integer> clients = servers.get(cs.getId()).getClients();
7      Iterator<Integer> i = clients.iterator(); //make an iterator for set
8      //close all client sessions before closing server connection
9      while (i.hasNext()) {
10         int id = i.next(); //get id of the client
11         CloseSessionMessage close = new CloseSessionMessage(id);
12         //sending close session message to the client
13         observer.fromSessionManager(close, id);
14     }
15     //remove server from the map
16     servers.remove(cs.getId());
17     //send back close session (use in IO device)
18     observer.fromSessionManager(cs, cs.getId());
19 break;
```

This is a code example of the handling of a close session message from the server. A close session message is sent to all clients in the set and after that a server is removed from the maps. Very interesting line of code is four. Methods "deserialization" and "getBytes" are discussed next.

Method "**getBytes"** returning Byte Buffer contains bytes of the concrete message in a proper order. It was created because sometimes it is required to use specific methods of the given class. It can be discussed next why polymorphism is not used instead of this.The answer is obvious, since it is impossible to put all bodies of the possible required methods to the parent and then inherit them. Thanks to this method bytes of the concrete message can be gained, but having bytes without any meaning is useless.

Method "**deserialization"** solves this problem. It needs the return value of the "getBytes" method like a parameter and it tags together all attributes from bytes.

Thanks to the knowledge of the order of parameters inside bytes, it is easy to build back attributes. Then get methods of the concrete type of the message can be used.

## 6.7 Final version of the GW

Figure 9 shows the complete GW after all phases. All modules and important methods can be seen there. From the main method is created GW observer (1) which creates Device IO module and Session Management module (2). Servers and clients are separated by different ports, so actually two threads are waiting for new connections (3). After connection is established a new thread is created which handles socket communication with the given client (4). Received message is sent by the method "onMessageCome" (5) to the GW observer and from there to the Session Management (6). Here is a message linked with the correct pair or special type of messages handle, like open or close session messages (7). Afterwards a message is sent back to the GW observer by the method "fromSessionManager" (8). Then a message is posted to the Device IO (9) where a correct thread is found and a message is sent to the socket (10).

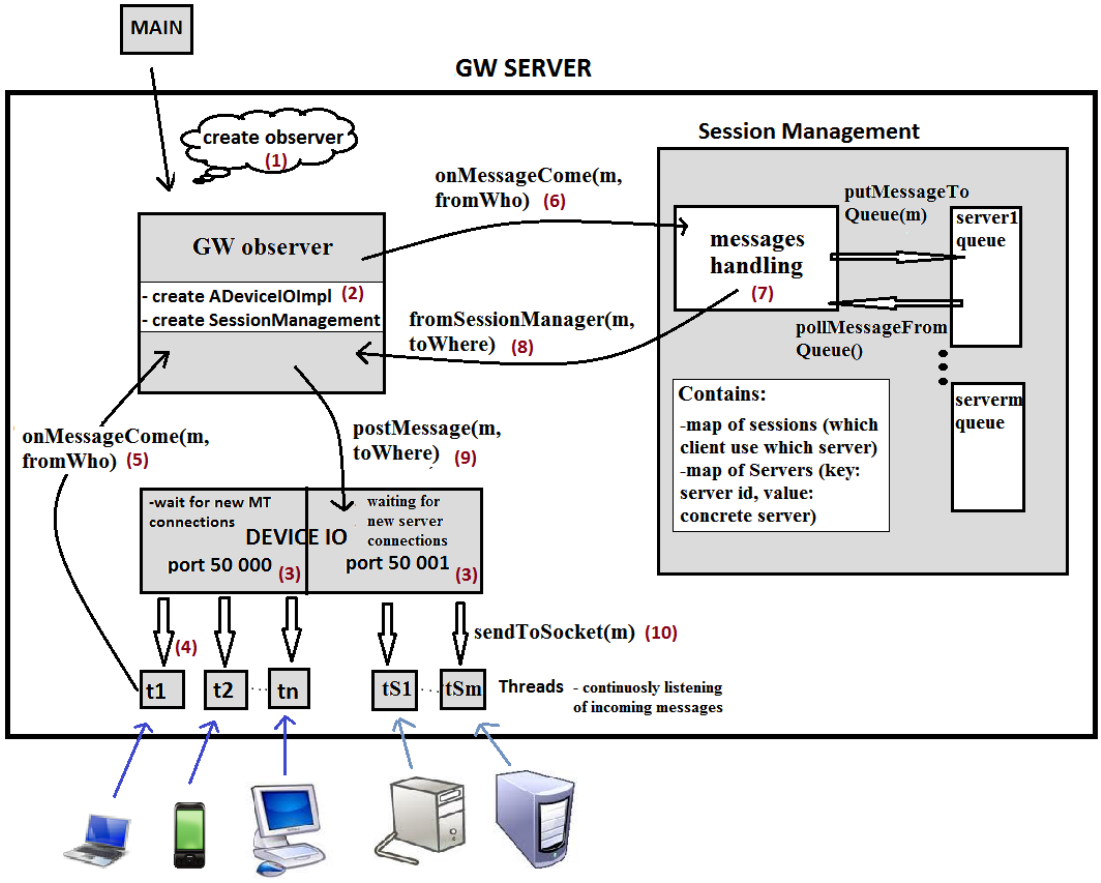


FIGURE 9. Final version of the gateway

# 7 SENDING STRINGS AND FILES IN JAVA AND OTHER PLATFORMS

## 7.1 Sending and receiving 32b integers in other platforms

It was described before how to send and receive integers in the Java gateway. A number was divided into four bytes and written to the socket, while receiving it is an opposite procedure. Bytes are put together and the result value is 32b integer.

If all platforms do the same procedure, it is secured that same values can be worked with anywhere. If using an Android device it is practically the same as in GW because Java language is used for programming. How it looks in Qt C++ or Python is discussed next.

### 7.1.1 Qt C++

Similar inheritance to Java can also be used in C++ language. A"Message" class with methods "writeInt32b" and "readInt32b" can exist there. First **sending 32b integer** out is discussed. A char array is set to proper values and sent out. Four bytes are sent at one time, not one by one like in Java. Of course, it is possible to send them one by one, but this looked like better solution. A code example can be seen below.

```
1   // converts inserted integer into 4bytes char array
2   void Buffer::setBuffer(int value) {
3       this->bufOut[0] = ((value >> 24) & 0xFF);
4       this->bufOut[1] = ((value >> 16) & 0xFF);
5       this->bufOut[2] = ((value >> 8) & 0xFF);
6       this->bufOut[3] = (value & 0xFF);
7   }
8   //method for writing 32b int into the socket
9   void Message::writeInt32b(QTcpSocket *socket, int value) {
10      buff.setBuffer(value); //set char buffer for sending proper int
11      qint64 bytesNumberOut = socket->write(buff.bufOut, sizeof(buff.bufOut));
12      socket->flush();
13  }
```

In the method "setBuffer" bitwise operators are used to set bits of bytes to proper values. And in the "writeInt32b" these bytes are just sent to the socket. A pointer to the socket is passed like a parameter together with a value that should be sent. On the line 11 is shown how to send more bytes at once. A variable "bytesNumberOut" can be used for r debugging. It is assigned to it how many bytes have been written.

Reading in Qt caused for major problems in the project. It was tried to make a thread for constantly reading. Resolving this problem was time-consuming, because the program crashed when sending some data out of thread. Then it was realized that all socket handling has to be inside a thread. Then a signal that alerts that bytes are ready to read did not work. Another signal “readyRead()” and no thread had to be used then. Thus, in Qt no method was found that waits until some bytes come. This signal illustrates this and some slot can be connected to that handles received bytes. An example is shown in the code.

```
1  //connect signal to the slot
2  QObject::connect(&socket, SIGNAL(readyRead()), this, SLOT(startRead()));
3  ...
4  //inside a startRead slot
5  lineLength = socket.read(buf, sizeof(buf));
6  //buf was divided to 4B
7  ...
8  // converts 4bytes char array into integer
9  int Buffer::getBufferInt() {
10     quint8 i0,i1,i2,i3;
11     i0 = bufIn[0];
12     i1 = bufIn[1];
13     i2 = bufIn[2];
14     i3 = bufIn[3];
15     return ((i0 << 24) | (i1 << 16) | (i2 << 8) | (i3));
16 }
```

Maybe it is not obvious but the most important line is number ten. It was very important to define partial results like “quint8”. It means that we use unsigned 8 bits int. This is the same with all platforms supporting Qt framework. If normal “int” is used, a wrong minus value is obtained.

### 7.1.2 Python

To learn the basics of Pythod took the author a relatively short time and after few days he was able to design classes and methods. It was found out how to make a thread and make unsynchronized communication. Basically several things can be coded very easily and in fewer lines of code than in Qt.

In the code example below sending an integer is illustrated.

```
1  def send(self,value):
2          convertValue = struct.pack("!i",value)
3          self.socket.send(convertValue)
```

A “struct” was imported and it automatically packs the integer value to four bytes. In Python method “send” is used instead of “write” from Java. It is basically able to send everything.

Receiving integer value is also simple and the code example is shown below.

```
1  def receive(self):
2          data = self.socket.recv(1)
3          data = data+ self.socket.recv(1)
4          data = data+ self.socket.recv(1)
5          data = data+ self.socket.recv(1)
6          myint = struct.unpack("!i", data)[0]
7          return myint
```

Method “recv” is used to receive a byte from the socket. Then bytes to an integer value are unpacked and returned.

## 7.2 Sending and receiving strings

Until now all communication was based on sending numbers inside messages. Sending some strings is also a very important part. This type of message is needed very often. Sending strings can be a topic for one thesis because many issues are connected with that in particular if other than ASCII values are required. Luckily just ASCII strings were used, where for one character one byte is required (255 possible characters).

### 7.2.1 Java GW

Every time when a new type of message was required, the GW had to be started with. It is mediator between others and it has to work firstly there. Firstly sending and receiving just some bytes array was tried. Sending bytes array showed up like a good solution. For receiving method “readLine” was rather used. This method receives bytes while it finds some of these symbols: ‘\n’ or ‘\r’. It returns a line of text without those ending symbols. It was agreed that all platforms have to send also ‘\n’ char at the end of the message. Below are examples of “write” and “read” methods from some Message classes that send and receive also strings. There is illustrated the sent type, client’s id, length of the string and string itself.

```
@Override
public void write(OutputStream out) throws Exception{
        super.write(out);
        writeInt32b(out, this.id);
        writeInt32b(out, length);
        //get bytes from the message
        byte [] bytes = message.getBytes();
        out.write(bytes);
}
```

```
@Override
public void read(InputStream in) throws Exception{
        super.read(in); //reading first integer values
        id = readInt32b(in);
        length = readInt32b(in);
        BufferedReader inString = null;
        inString = new BufferedReader(new InputStreamReader(in));
        message = inString.readLine() + "\n"; //pasted also end char '\n'
        length = message.length();
}
```

### 7.2.2 Qt C++

Thanks to the new features provided by Qt it is far more easy to work with strings than in the pure C++. A “QString” class provides basically similar methods to “String” class in the Java. Sending a string message was again an easier task then receiving it. “QByteArray” was used to put “QString” like a bytes array. An example of this is shown below:

```
QByteArray block; //block of bytes with undefinate lenght
block.append(message); //append QString message into
block.append("\n"); //append end char
socket->write(block); //write block to the socket
socket->flush(); //flush content
```

Reading of the string is made similar to reading integers. It is read byte after byte until a length of the string message is reached that t was sent by the attribute length. A method “push_back” of the class “QString” was used. It pushes the byte to the end of the string.

### 7.2.3 Python

In Python the situation is again easy. The same functions “send” and “receive” can be used again. A small part of code is shown below.

```
def sendString(self, message):
    msg = message + '\n'
    self.socket.send(message)

def receiveString(self, len):
    message = ""
    message = self.socket.recv(len-1)
    self.socket.recv(1)  #enter
    return message
```

To send a string is simple: it is to be sent just as it is. It is possible to send any ASCII character. It is better to be aware of using this character: '\'.

Length – 1 bytes of the string is read at the receiving. It is because '\n' character is not needed. It is very important to read also that last character because it will be read in next expecting value, for example in type reading. This causes some major problems in the project at first. After receiving a string all types came wrong. It was all because of that one byte.

## 7.3 Sending and receiving files

Sometimes it is needed to send also some files like images, mp3s and text files. One task was to create a special type of message for sending small files. If looking at the file it is just a stream of bytes. Any type of the file can be open like some input stream of bytes. Theoretically bytes of file were needed and sent like any other bytes before. A problem is that a file can consist of thousands of bytes. Because of that the whole file is not received at once but bytes of file are fragmented into smaller packets.

### 7.3.1 Sending and receiving files in the Java GW

The idea of the gateway in this case is to send bytes of a file through from one client to another without any saving. Testing the configuration was extremely important in this case and it is discussed later in more detail. This task caused more troubles than was thought. Sending a file message contains these attributes: type, id of client, size of the file, length of file name, file name and bytes of file. Thus, this message is huge compared to others.

A test client has to open a file and get bytes from the file input stream. First the user is to set all attributes and then use a "write" method from "SendFileMessage" class. This method is not too complicated. All attributes are sent like in previous types of

messages. Then bytes of file are sent with the help of data output stream. Method "write" can be overloaded and also array of bytes sent.

```
1   //send all attributes
2   DataOutputStream dos = new DataOutputStream(out);
3   dos.write(bytesOfFile);
4   dos.flush();
```

Receiving is again more complicated. As described the whole file is usually not sent at once. It is divided into more packets and because of that a more sophisticated reading is needed. Bytes of file read inside a loop until bytes read = size of file.

```
1   bytesOfFile  = new byte[size]; //allocate byte array with size of file
2   int bytesRead = ins.read(bytesOfFile, 0, size); //try to read as much as possible
3   int current = bytesRead; //current store bytes read until now
4   if (current < size)
5       do {
6               //try to read rest of file
7               bytesRead = ins.read(bytesOfFile, current, (size-current));
8               if(bytesRead >= 0) //if it was read something
9                   current += bytesRead; //increase current by bytes read
10          } while(current != size); //read until all bytes of file are read
```

It does not look like there were some troubles. But to find this solution took a lot of time. One small sleep is needed between sending attributes of the send message file and bytes of the file. Without that sleep not all bytes come and the method "read" stacks in one place. This was really an issue that could not be resolved in the given time.

### 7.3.2 Files in Python

When it is required to send a file in Python, the file is opened in a binary read mode. Then the whole file inside variable can be read and afterwards the file can be sent. In Python it is relatively easy to send a file.

```
1   def sendFile(self):
2           fileToSend = open(self.file_name,"rb") #open file
3           filee = '' #prepar variable
4           filee = fileToSend.read() #read bytes of the file
5           self.socket.send(filee) #send bytes to the socket
```

When a file is receiving, at first a new file must be created and opened for writing binary files. Receiving a file requires also making some kind of loop for reading bytes until the whole file is received. The loop is disturbed if there are no more bytes to read. After finishing reading, the file is closed.

```
def receiveFile(self):
        dst = open(self.file_name,"wb+") #create a file
        filee = '' #prepare variable
        while True:
            filee = self.socket.recv(self.size) #rcv file
            if not filee: break #if rcv ends, end reading
            dst.write(filee) #add bytes to the file
        dst.close() #close the file
```

# 8 ASYNCHROUNOUS COMMUNICATION AND THREADS IN ANDROID

## 8.1 Threads in Android

“When an application is launched, the system creates a thread called "main" for the application. The main thread, also called the **UI thread**, is very important because it is in charge of dispatching the events to the appropriate widgets, including drawing events. It is also the thread where your application interacts with running components of the Android UI toolkit. When the application performs intensive work in response to user interaction, this single thread model can yield poor performance. Specifically, if everything is happening in the UI thread, performing long operations such as network access or database queries will block the whole UI. When the thread is blocked, no events can be dispatched, including drawing events. From the user's perspective, the application appears to hang. After five seconds a force close message is shown and the actual process is closed”. This is of course inappropriate and a user of mobile phone would uninstall such a poorly programmed and freezing application. (http://developer.android.com/resources/articles/painless-threading.html)

“To perform a long lasting operation it is good to create a working thread that can work in the background and UI thread is not blocking and so the user can still interact with the components on the widget. Basically it is possible to create threads like in the classical Java. It should be remembered that the Android UI toolkit is not thread-safe. It means that all manipulation with the user interface must be performed from UI thread. If that is tried from the working thread, an application shows a force close. Thus, there are simply two rules to Android's single thread model”: (http://developer.android.com/resources/articles/painless-threading.html)

1. Do not block the UI thread.

2. Do not access the Android UI toolkit from outside the UI thread.

It is possible to create a thread anywhere and then use one of these methods to pass data from the working thread to the UI thread:

* Activity.runOnUiThread(Runnable)

* View.post(Runnable)

* View.postDelayed(Runnable, long)

This approach can be very difficult to maintain code. To handle more complex interactions with a worker thread, using a Handler was considered in a worker thread, to process messages delivered from the UI thread. Another option can be to use "AsyncTask" class. Both options are explained in the next subchapters.

### 8.1.1 "AsyncTask" class

There are a few threading rules that must be followed for this class to work properly:

* The task instance must be created on the UI thread.

* execute(Params...) must be invoked on the UI thread.

* Do not call onPreExecute(), onPostExecute(Result), doInBackground(Params...), onProgressUpdate(Progress...) manually.

* The task can be executed only once (an exception will be thrown if a second execution is attempted.)

This class is a very easy way to perform threading in Android. AsyncTask allows to perform asynchronous work on the user interface. It performs the blocking operations in a worker thread and then publishes the results on the UI thread, without requiring us to handle threads and handlers ourselves. It provides many useful methods to communicate with UI thread and allow also drawing changes in main UI. It can be used since Android 1.5. (http://developer.android.com/guide/topics/fundamentals/processes-and-threads.html)

To use this class a new private class is needed that inherits "AsyncTask". A method "**doInBackground()**" has to be implemented which runs in a pool of background threads. An example below illustrates this (retrieved from:

```
public void onClick(View v) {
    new DownloadImageTask().execute("http://example.com/image.png");
}

private class DownloadImageTask extends AsyncTask<String, Void, Bitmap> {
    /** The system calls this to perform work in a worker thread and
      * delivers it the parameters given to AsyncTask.execute() */
    protected Bitmap doInBackground(String... urls) {
        return loadImageFromNetwork(urls[0]);
    }

    /** The system calls this to perform work in the UI thread and delivers
      * the result from doInBackground() */
    protected void onPostExecute(Bitmap result) {
        mImageView.setImageBitmap(result);
    }
}
```

This code presents an example of a typical situation when it is good to use "AsyncTask" thread. A picture from the internet should be downloaded. Without a thread it can cause freezing of the application.A "DownloadImageTask" private class is created which inherits "AsyncTask". From the main UI method "execute" is called with the URL address of the image. This method calls "doInBackground" method which is the thread for downloading. A result is passed to the method "**onPostExecute**" and this method can communicate with the main UI thread and changes screen.

An asynchronous task is defined by 3 generic types, called Parameters, Progress and Result, example: AsyncTask<String, Void, Bitmap>. Parameter more strings can be passed and they can be resolved one by one. There are more useful methods that can be overridden.

**"onPreExecute"** method is called before "doInBackground" method is called and it can for example show up a progress dialog.

**"onProgressUpdate (Progress... values)"** Runs on the UI thread after publishProgress(Progress...) is invoked. The specified values are the values passed to publishProgress(Progress...). It can be for example refreshed progress dialog shown in main UI.

**"publishingProgress(Progress…)"** This method can be invoked from "doInBackground(Params...)" to publish updates on the UI thread while the background computation is still running.

As can be seen, this class provides an easy way to use thread for some background activities. Progress is easy to show and return results can be returned into the main UI thread.

### 8.1.2 Using “Handler” class

Although an “AsynTask” looks like a universal solution for threading in Android, sometimes another approach is needed for making threads. A data can be returned from “AsyncTask” after a thread is finished or through publishing progress. Thus if continuous listening is needed, it can be better to write the developer’s own handler for receiving data from the thread.

Inside “Handler” class method “**handleMessage(Message msg)**” should be rewritten. From the parameter “msg” can be obtained data and then specific basic types like int, byte according to the name of parameter passed from thread. Now a private class can be written which inherits from Thread. A handler instance can be passed through the constructor. Then all what is needed is to put some data to the Bundle and pass them to the message. Then method handler.**sendMessage(msg)** can be used to send data to the Handler. Inside handler can be decided what should happen with the income message. The data can be used in the main UI thread.

## 8.2 Asynchronous communication using handler

It is better to make an own handler and thread that continuously listens to incoming messages. Without this it is possible to make only synchronized communication. There is a problem that only one type of a message is expected and it calculates only with no problem case. But what if the server or gateway sends for example a close session message and some other message is being expected? Or some file is needed to receive. It causes freezing of the application and the synchronized communication is not as large a problem as freezing.

It is good to use a singleton class for handling socket communication. In this class Handler is written and a private class extends thread. A code example of the Handler is shown below.

```
1  final Handler handler = new Handler() {
2      public void handleMessage(Message msg) { //rewrited method
3          //get type from data passed from thread
4          int type = msg.getData().getInt("type");
5          switch (type)
6          {
7              case MessagesIDConstantsInterface.CLOSE_SESSION:
8                  CloseSessionMessage cs = new CloseSessionMessage(id);
9                  sendMessage(cs); //send message out of handler
10                 break;
11         }
12     }
13 };
```

A method “handleMessage” is called from the thread after “sendMessage” was called. On the fourth line is shown how some data from the Message can be obtained. In the switch block there are all messages that can be received. In this code example there is only one message because its purpose is only to give an overview how to make an own handler.

The handler is passed like a parameter to the new private class extending Thread. Here is overridden method “run”. This is called from “SocketHandler” by calling “start” method. Inside “run” is infinite loop for listening incoming messages similar to the GW.

```
1  public void run() {
2      running = true;
3      while (running) {
4          Message msg = handler.obtainMessage(); //get Message from handler
5          Bundle bundle = new Bundle(); //create a new Bundle
6          try{
7              int type = Message.readInt32b(in); //read type of the message
8              switch (type) { //switch according to the type
9                  case MessageIDConstantsInterface.CLOSE_SESSION:
10                 CloseSessionMessage cs = new CloseSessionMessage();
11                 cs.read(in); //read message from the socket
12                 bundle.putInt("type", MessageIDConstantsInterface.CLOSE_SESSION);
13                 msg.setData(bundle); //set data of the Message
14                 handler.sendMessage(msg); //send data (Bundle) to the Handler
15                 break;
16                 ...    //other types of messages are ommited
17             }
```

On lines 13 and 14 is shown how to send some bundle data to the Handler. It is important to use the same name for passed attributes inside the Thread where data are set and inside the Handler where data are got (see the fourth line from previous code example and 12$^{th}$ line in this code example). Thanks to this approach asynchronous communication is secured between Android device and GW. For writing messages to the socket “OutputStream” is got everywhere in the application.

# 9 TESTING

## 9.1 How to test

One option is to test the application with real devices and servers. This is of course needed but this approach is inappropriate when the software is in its first phases. Why is it so? Testing on real devices is time consuming and it requires ready and functional software running on all clients. Gateway is divided into few modules that can also be tested separately.

Because of that a test module must be developed that behaves like a real server or client and can be run in the same machine. For this purpose a "TestClient" class was created to represent servers and also clients. It contains a main method, so when GW application is run a second option appears, to run the test client. Whereas this class is a part of whole application it can use its classes and methods. This is very useful because a new application does not have to be created and all needed classes copied but it is used and tested in the actual state of the program. A "TestClient" class uses"Client" and "Server" classes. There are methods like send open session, receive it, send file, receive file etc. These methods are used in "TestClient" where it is possible to change running configuration to any possible one.

The testing was an important part from very first phases. Every described step in the *Design* chapter was properly tested. After successfully running several testing configurations, a module or a new component was considered to be functional. Then it was safe to build further functionality on good bases. Many times it happened that when truing the whole application on real devices and something crashed it was a problem somewhere else than on GW. Thanks to extensive testing of GW it was also easier to find the problems in other parts of the system.

## 9.2 Console debugging

The easiest way to watch what happening while the program is running is to use a console printout. It is important to print out only important information, otherwise a printout is chaotic. It is also good to write where the printout took place and for what reason. In Java system library is used and it can look like this: "*System.out.println("GW started to work");*" A console output is usable also when a

program is run from Linux console. But if there are more clients and servers connected and they communicate together, a console output is unclear and it is extremely difficult to find some older situation when something failed. Because of that it is good to use also logging to file.

## 9.3 Logging to file

Logging provides a clear output in a text file that is practically unlimited. Console output can show just a limited number of lines. When the GW is running for a longer period of the time and during its operating something crashes, a programmer can copy a log file and find out what happened. Every line contains also time and date, so it is easy to find a bug. Afterwards a programmer can analyze a problematic part of the code or a situation that involved it.

A "LogFile" class was created to handle logging to the txt file. It has public static methods for writing to the file. It means that these methods can be used without creating an instance. From any part of the program it is possible to write into the file. A programmer decides what should be logged in. In the GW application only important events and error tip-offs are logged in.

```
1  public static void write(String file, String logMessage) throws IOException
2  {   //set up correct time formating
3      TimeZone tz = TimeZone.getTimeZone("GMT+2:00"); // set up proper time zone
4      Date now = new Date();
5      DateFormat df = new SimpleDateFormat ("dd.MM.yyyy hh:mm:ss ");
6      df.setTimeZone(tz);
7      String currentTime = df.format(now);
8      //write a log message into the file
9      FileWriter aWriter = new FileWriter(file, true);
10     aWriter.write(currentTime + " " + logMessage + "\n" + "\r");
11     aWriter.flush();
12     aWriter.close();
13 }
```

In the code example above is shown how to set correct formatting of the time. On the seventh line formatted date is assigned to the String. On line 10 is "write time and log message into the file" with end characters for the next line inside the file. In the notepad used in Microsoft's Windows these characters do not work.

## 9.4 An example of the testing configuration

A brief look is presented to one of possible configurations. In this configuration just one server and one client are used for an easier orientation in the console debugging.

At first is run gateway and it wrote that it is waiting for new connections. After that test client with this configuration is run:

- firstly is send open session from the server
- client connects to the GW and receives ID
- client sends open session message and the server receives it
- client sends message with integer value, server receives it and sends back
- client reads an integer message
- client sends text message: "*Hello server*!"
- server receives message and sends it back to the client
- client sends file ("*ddms.png")* to the server and server stores it like "*copy_ddms.png*"
- server sends file ("*1.jpg*") to the client and it stores it like "*copy_1.jpg*"
- client sends close session and the server receives it
- server sends close session to the GW and test client finishes

This simple configuration shows that all mentioned types of messages really run between client, gateway and server in both directions. "TestClient" is prepared to insert more clients and it is also easy to add another server. Then it can be made surethat linking messages between proper clients and servers are functional also.

In the figure below is shown a console output in terminals. A terminal on the left side is output from GW and on the right is output from "TestClient".

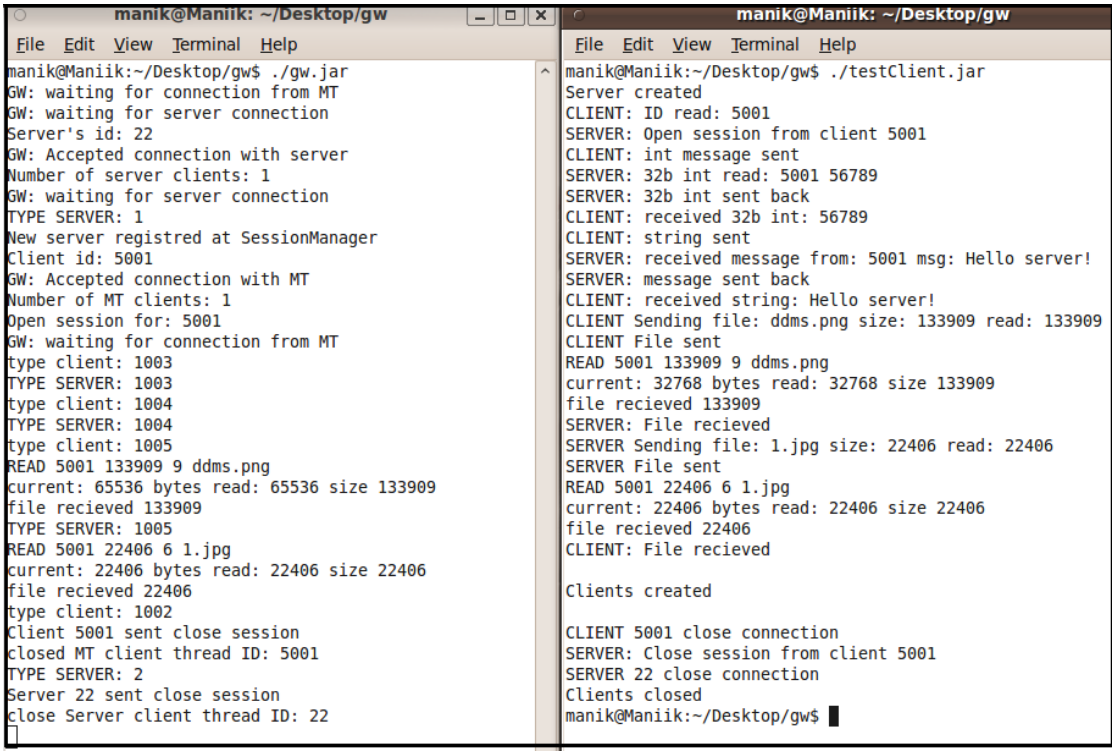


FIGURE 10. Console debugging on specific running configuration

All important events that happened can be seen. At the console output on the left is shown registering on the GW, accepting connections with server and client and types of messages send between them. At the end a client sends close session and also server.

At the console output on the right is shown the sent integer message from a client with value 56789 and the same value was read in the server and afterwards also back in the client. Then string message "*Hello server!*" is sent and it can be seen that it is the same in server and also the same after return back. Finally is sent file "*ddms.png*" from the client to the server and file "*1.jpg*" from the server to the client. It can be checked that in the folder there are really new files "*copy_ddms.png*" and "*copy_1.jpg*".

GW was tested also with real devices in the testing environment. In all demonstrations GW was reliable, stable and it was a solid core of the whole environment. Only small problems were encountered with sending files.

# 10 PROJECT EVALUATION

## 10.1 Achievements

All critical tasks and requirements were accomplished. Communication between more platforms was reliable and stabile. It was tested both in virtual and in the real environment.

It was possible to send numbers, strings and files between clients and combine them according to needs. GW was made in the way that it is really easy to add new types of messages. If it is required, whole modules can be replaced by others with more advanced technologies, for example rebuilt “DeviceIO” module from threads to server socket channel. This would be needed if an internal project should be changed to a real customer project. Then also new security and encryption issues would appear.

## 10.2 Scrum - personal experiences

Since this was my first working experience, my first sprint planning meeting was a big challenge. It was the first time we familiarized ourselves with a concept and idea. We prioritized the whole product backlog. Of course we should start with some core tasks and therefore it was an important part of our session. The next step was to estimate the amount of the time needed to finish everyone’s task. It was a difficult task for us because we did not know most of the technologies. We used a very interesting method called poker planning. Everyone had cards with numbers 1, 2, 3, 5, 7, 11, 23 and infinity or question mark. After small discussion about the content of the task we prepared a card with some value representing time. Then we showed cards and compared them. If everyone from the team showed a card with a neighbor’s value, we chose the bigger value. If we had more values, a member of the team with the smallest value had to explain why he thinks it would take so little time and the same for the member with the highest value. Then we voted again. After this process we made the sprint backlog.

We used JIRA software for managing the Scrum process. Every member of the team chose one task from the task board and implemented it. It is possible to edit tasks, for example to add new subtasks if needed. Every day we had a daily Scrum where we talked about our progress or problems and after that we solved them. Our sprints were

usually two weeks long. We used to perform a demo every second Friday where the product owner could see our progress.

Scrum methodology was very effective and we soon made progress and our skills in estimation of the time increased. Very often we had to accept new tasks and we implemented them inside project. Thanks to flexible Scrum it was no problem.

## 10.3 Conclusion

A practical training in the Ixonos was for me my first job experience. I wanted to be a good employee who can work individually without any special help. I wanted to satisfy my employer's expectations and be proud of my work. I expected that I had to learn many new things and use them immediately in the project.

I reached all mentioned goals and I am happy that my employer was satisfied with my work and also with our whole team. We made all demos what were required and it is possible that they will change to real customer projects. I had free hands in the development process, so I could realize my own ideas and do things in my way. I had to accomplish all tasks in reasonable time and if some problems occurred, my colleagues helped me.

I learnt many new things and this practical training was the best opportunity how to connect the learnt things from the school with real work. I extended my knowledge in Java, Android, Qt languages. I learnt a new Python language and I familiarized myself with a great number of new terms from my field of study. I saw also other technologies and languages with which my colleagues worked.

Working in the Ixonos company was great. I worked in very interesting projects, using new technologies and ideas. I had great colleagues; communication with them was always interesting. I improved my language skills and communication skills. Working on this thesis helped me find out a new ways of expressing myself and improved my writing skills. I obtained self-confidence. I had an opportunity to cooperate with the leading software architect of the Ixonos company.

# REFERENCES

Cay S. Horstmann and Gary Cornell, September 2007, Core Java: Volume I, Fundamentals (8th Edition), Publisher: Prentice Hall, ISBN: 0-13-235476-4

Android Developer's Guide, referenced May 2011
http://developer.android.com/guide/index.html

Java SE Technical Documentation, referenced May 2011
http://download.oracle.com/javase/

Qt Reference Documentation, referenced May 2011
http://doc.qt.nokia.com/4.7-snapshot/index.html

The Java Tutorials, referenced May 2011
http://download.oracle.com/javase/tutorial/

Wikipedia's Data Structure Article, referenced May 2011
http://en.wikipedia.org/wiki/Data_structure

Wikipedia's Hash Table article, referenced May 2011
http://en.wikipedia.org/wiki/Hash_table

Wikipedia's Internet Socket article, referenced May 2011
http://en.wikipedia.org/wiki/Internet_socket

Wikipedia's Queue (data structure) article, referenced May 2011
http://en.wikipedia.org/wiki/Queue_%28data_structure%29

Wikipedia's Set (computer science) article, referenced May 2011
http://en.wikipedia.org/wiki/Set_%28computer_science%29

Wikipedia's Thread (computer science) article, referenced May 2011
http://en.wikipedia.org/wiki/Thread_%28computer_science%29